กีรติ พิฑูรพงศ์

หน่วยความจำ

พื้นฐานของ X86

กล่าวถึงการใช้งานและวิธีการต่าง ๆ ของหน่วยความจำ ที่เป็นพื้นฐานของ X86

การใช้งานหน่วยความจำคอมพิวเตอร์ที่ใช้ซีพียูตระกูล X86 เริ่มต้นด้วย 286, 386, 486 ในส่วนที่ต่ำกว่า 1 เมกะไบต์ มีลักษณะคล้ายกัน สำหรับท่านที่พอจะรู้เกี่ยวกับภาษาแอสเซมบลีและซีมาบ้างแล้วก็จะดีมาก แต่ถ้ายังไม่เคยรู้จักมาก่อนเลยผมก็จะกล่าวถึงบางเรื่องเท่าที่จำเป็น หรือศัพท์ทางด้านคอมพิวเตอร์และทางตรรกศาสตร์บ้างเล็กน้อย เพื่อช่วยให้ท่านเข้าใจได้ดีขึ้น ถ้าท่านมีหนังสือเกี่ยวกับ ซีพียู 80X86, IBM PC, Digital, ฯลฯ อ่านประกอบไปด้วยก็จะช่วยให้ท่านเข้าใจได้ดียิ่งขึ้น ดังนั้นผู้ใดที่รู้สึกซึ้งแล้วจะอ่านบทความนี้เล่นแก้กลุ้มก็แล้วแต่ท่าน แต่ถ้าท่านยังไม่ทราบหรือยังงงอยู่บทความนี้อาจช่วยได้นะ..จะบอกให้

พื้นฐาน

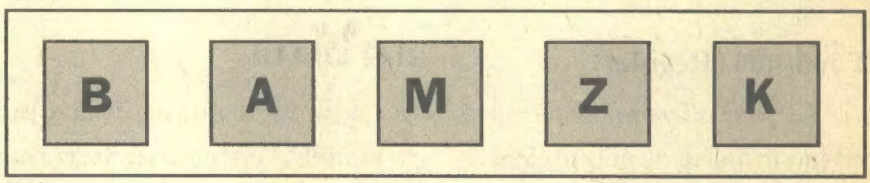
1. แอดเดรส (address)

สมมติว่าผมมีกล่องอยู่กล่องหนึ่ง โดยที่กล่องชนิดนี้สามารถใส่ตัวอักษรได้ครั้งละ 1 ตัวเต็มพอดี และถ้าผมมีกล่องแบบนี้อยู่ 5 กล่อง และผมนำตัวอักษรไปใส่ไว้ในกล่องเหล่านั้นทุกกล่องดังรูปที่ 1

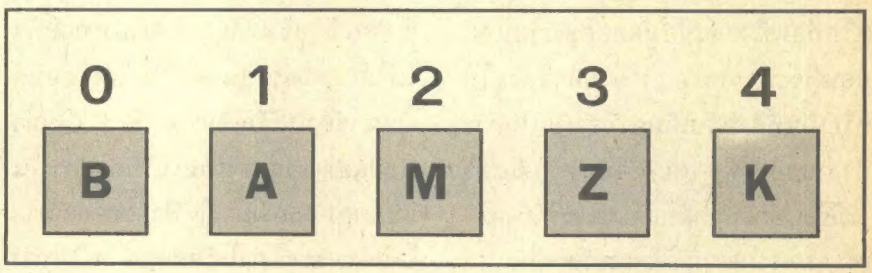
และแล้ว..วันหนึ่งผมเกิดคิดถึงน้อง A ขึ้นมา ผมจึงสั่งให้เพื่อนช่วยหยิบกล่องใบที่ผมใส่น้อง A เอาไว้ "ใบไหนล่ะ?" "ก็ใบที่สองจากทางขวาไง" "ขวาของฉันหรือของคุณ" "ขวาของคุณ" วุ่นวายดีแท้ แต่ถ้าผมนำหมายเลขไปติดเอาไว้ที่กล่องแต่ละใบดังรูปที่ 2

"ช่วยหยิบกล่องหมายเลข 1 ให้ที" ง่ายกว่ากันเยอะเลย

ผู้ที่ออกแบบซีพียู X86 เขาให้ X86 มีกล่องคล้าย ๆ แบบนี้ได้ล้านกว่าใบ (2^{20}) กล่อง แต่ละใบของเขาใส่ข้อมูลได้ขนาด 1 ไบต์ ดังนั้นถ้าจะกล่าวว่า X86 มีเมมโมรีได้ถึง 1 เมกะไบต์ ก็



รูปที่1



รูปที่2

น่าจะได้ และหมายเลขที่ติดไว้ที่กล่องแต่ละใบเขาเรียกว่า แอดเดรส ดังนั้นถ้าต้องการนำข้อมูลออกมาหรือใส่เข้าไปที่เมมโมรีก็ต้องระบุแอดเดรสของเมมโมรีเอาไว้ด้วย โดยที่แอดเดรสจะเริ่มจาก 0 ไปจนถึงล้านกว่า แต่ในการกล่าวถึงจำนวนใด ๆ ในทางคอมพิวเตอร์มักจะอยู่ในรูปของเลขฐานสอง ฐานแปด หรือฐานสิบหก ถ้าจะกล่าวถึงแอดเดรสเป็นฐานสิบหกก็น่าจะดูดีกว่า และถ้าผมจะบอกว่าแอดเดรสเริ่มต้นจาก OH ไปจนถึง OFFFFFH ผมก็ไม่ผิดใครจะทำไม..

ค่าแอดเดรสที่กล่าวไปนั้นมีขนาด 20 บิต ไม่เชื่อลองทำดู และเขาซึ่งเป็นใครก็ไม่ทราบเรียกแอดเดรสขนาด 20 บิตของผม (ที่จริงเป็นของ X86 ที่ intel เขาผลิต) ว่าฟิสิคอลแอดเดรส (physical address) ซึ่งก็เหมือนเป็นการอธิบายว่ามีเมมโมรีจริง ๆ ที่เมื่อเปิดฝาเครื่องคอมพิวเตอร์ออกแล้วจะเห็นว่ามีเมมโมรีอยู่ 1 เมกะไบต์ อย่าสับสนว่าเมมโมรีเหล่านี้อยู่ในซีพียู X86 เป็นอันขาด เนื่องจากมันเป็นเมมโมรีที่อยู่ในรูปของ IC (หรือในอนาคตอาจอยู่ในรูปแบบอื่น ๆ) ที่ผู้ผลิตคอมพิวเตอร์ต้องนำมาต่อเพิ่มภายนอกเอง

2. รีจิสเตอร์ (Register)

คงจะไม่ผิดนักถ้าหากจะกล่าวว่ารีจิสเตอร์ก็คล้าย ๆ กับเมมโมรีขนาดเล็ก ๆ ที่อยู่ภายในซีพียู โดยที่ซีพียูจะใช้รีจิสเตอร์เป็นที่พักข้อมูลจากภายนอกหรือก่อนที่จะส่งข้อมูลออกสู่ภายนอกก่อนที่จะทำการใด ๆ ตามคำสั่งต่อไป หรือเป็นที่สำหรับเก็บค่าที่จำเป็นในการทำงานของซีพียู โดยที่รีจิสเตอร์แต่ละตัวจะมีชื่อเรียกแทนคล้ายกับเมมโมรีที่มีแอดเดรส ผมจะกล่าวถึงรีจิสเตอร์เท่าที่จำเป็นเท่านั้น ถ้าผมกล่าวโดยละเอียดก็จะผิด

วัตถุประสงค์ของบทความนี้ไป (ที่จริงผมกลัวจะถูกจับผิดนะ..!?)

3. Shift

เป็นการเลื่อนบิตไปทางซ้ายหรือขวาแล้วได้ผลลัพธ์ใหม่ที่คล้ายกับการคูณหรือหาร เช่น ผมต้องการ shift ค่า 64, 40H (ฐานสิบหก) หรือ 1000000B (ฐานสอง) ไปทางซ้าย 1 บิต จะได้ค่า 128 หรือ 80H, 10000000B แต่ถ้า shift ไปทางขวา 1 บิต ก็จะได้ค่า 32, 20H หรือ 100000B จะเห็นว่าการเลื่อนบิตไปทางซ้าย 1 บิต จะคล้ายกับการคูณ 2 และถ้าเป็นการเลื่อนบิตไปทางขวา 1 บิต จะคล้ายกับการหาร 2 นั่นเอง

การใช้งาน

ในการใช้งานเมมโมรีเหล่านี้ในทางซอฟต์แวร์ เราไม่สามารถที่จะใช้งานเมมโมรีโดยใช้ฟิสิคอลแอดเดรสได้โดยตรง แต่จะต้องใช้รีจิสเตอร์ขนาด 16 บิต ที่ X86 มีอยู่จำนวน 2 ตัว มาประกอบกันให้ได้แอดเดรสขนาด 20 บิต สมมติว่าผมใช้ค่าในรีจิสเตอร์ A กับ B เป็นตัวบอกแอดเดรสของเมมโมรีที่ต้องการในแบบของ X86 ผมจะได้ฟิสิคอลแอดเดรสโดยการ shift ค่าในรีจิสเตอร์ A ไปทางซ้าย 4 บิต (ก็คือ คูณด้วย 16 นั่นเอง) แล้วบวกกับค่าในรีจิสเตอร์ B ก็จะได้ฟิสิคอลออกมา เท่านี้เองง่าย ๆ บางท่านอาจจะงงก็ไม่แปลกอะไร เพราะผมก็เคยงงเป็นเดือนเลย อาจจะยังไม่เข้าใจจะยกตัวอย่างให้ดูแต่เป็นฐานสิบหก (จะเป็นเลขฐานอะไร ก็คือ จำนวนเดียวกันนั่นเอง) ให้ A กับ B มีค่าเป็น 763BH และ 25BCH ตามลำดับ

shift A ไปทางซ้าย 4 บิต	763BOH
บวกกับรีจิสเตอร์ B	763BOH+ 25BCH
ฟิสิคอลแอดเดรส	7896CH

โดยที่จะเรียกค่าที่อยู่ในรีจิสเตอร์ A ว่าเซกเมนต์และค่าในรีจิสเตอร์ B ว่าออฟเซต ดังนั้นการอ้างถึงเมมโมรีในลักษณะนี้จะเขียนในลักษณะเซกเมนต์ต่อออฟเซตก็คือ A:B หรือ 763BH:25BCH และเรียกการใช้งานเมมโมรีเช่นนี้ว่า ลอจิกคอลแอดเดรส

ผู้ที่ออกแบบ X86 ได้กำหนดให้มีรีจิสเตอร์ที่เก็บค่าเซกเมนต์เพื่อใช้งานโดยเฉพาะคือ CS, DS, ES, SS ส่วนออฟเซตก็ใช้รีจิสเตอร์ตัวอื่นได้ตามลักษณะการใช้งานซึ่งจะกล่าวถึงต่อ ๆ ไป (ถ้ามีโอกาส) การใช้งานในรูปแบบของลอจิกคอลและออฟเซตที่แตกต่างกันได้ เช่น 763BH:25BCH กับ 7896H:000CH จะให้ค่าฟิสิคอลแอดเดรสเดียวกัน คือ 7896CH ดังนั้นจึงเป็นไปได้ที่จะเกิดการทับกันของข้อมูลถ้าหากใช้งานอย่างไม่ระมัดระวัง แต่ว่าในการเขียนโปรแกรมโดยทั่ว ๆ ไปนั้น โอกาสเช่นนี้เกิดขึ้นน้อยมาก ดังนั้นท่านจึงไม่ต้องระแวง

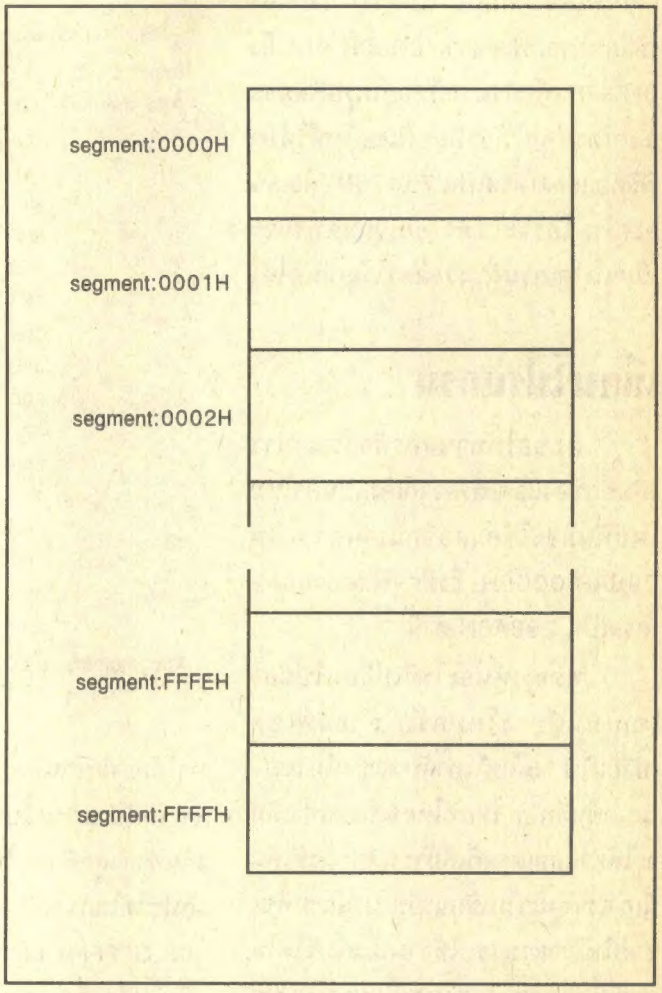
มีแค่ 64 กิโลไบต์เท่านั้น

ขณะนี้เราสามารถใช้เมมโมรีโดยระบุทางเซกเมนต์ต่อออฟเซตได้ ให้

ลองพิจารณาดูที่ออฟเซตจะพบว่าสามารถเริ่มต้นค่าได้ตั้งแต่ 0000H และถ้าเพิ่มค่าออฟเซตเข้าไปอีกหนึ่ง คือ 0001H ก็เท่ากับว่าเราระบุถึงเมมโมรีที่แอดเดรสถัดไปได้อีกเรื่อย ๆ เพียงแต่เพิ่มออฟเซตไปตามที่ต้องการ ดังรูปที่ 3

แต่อย่าลืมว่าออฟเซตของเรามีขนาด 16 บิต ดังนั้นค่าเต็มพิกัดของมันคือ OFFFFH ถ้าท่านเพิ่มค่าออฟเซตต่อไปอีกค่านี้จะวนกลับไปเริ่มที่ 0000H สรุปก็คือ ออฟเซตมีค่าได้เพียง 64K หรือกล่าวได้ว่าในหนึ่งเซกเมนต์เราสามารถเก็บข้อมูลได้สูงสุด 64 กิโลไบต์ บางท่านอาจเถียงว่าไม่จริงทำไมถึงคิดว่า OFFFFH+1 (บวกอีกหนึ่งไบต์จากออฟเซตที่ 0000H) เป็นฐานสิบแล้วได้ 65,536 ไบต์ ไม่ใช่ 64 กิโลไบต์ เนื่องจาก 1 กิโลไบต์ มีขนาด 1024 ไบต์ แล้วถ้าข้อมูลของผมหรือของคุณมีขนาดเกิน 64 กิโลไบต์ แล้วจะทำอย่างไร

รูปที่3



วิธีแก้อาการ

ก็ไม่เห็นจะยากอะไรเปลี่ยนไปใช้เซกเมนต์ถัดไปก็ได้..ดูเผิน ๆ เหมือนใช้ได้ แต่อย่าลืมว่าการใช้งานแบบลอจิกคอลแอดเดรสนั้น ถึงแม้ว่าจะเปลี่ยนเซกเมนต์ไปแล้วก็ตาม แต่ถ้าลองเปลี่ยนค่าให้ออกมาเป็นฟิสิคอลแอดเดรสแล้วอาจจะเป็นตำแหน่งเดียวกันกับเมมโมรีที่ท่านใช้เซกเมนต์ของเดิมอยู่ก็ได้ เช่น เดิมท่านเมมโมรีบริเวณ 6547H:0000H จนกระทั่งเต็มคือ 647H:FFFFH แล้วข้อมูลยังใส่ไม่หมด ท่านจึงเลือกใช้เซกเมนต์ถัดไปคือ 6548H โดยเริ่มออฟเซตที่ 0000H ลองคำนวณดูนะครับ

6548H:0000H เป็นฟิสิคอลแอดเดรส ได้ 65480H

ถ้าลองใช้เซกเมนต์ของเดิมคือ 6547H และออฟเซตที่ 0010H

6547H:0010H เป็น ฟิสิคอลแอดเดรส ได้ 6548H

จะได้เมมโมรีที่แอดเดรสเดียวกัน ผลก็คือ ข้อมูลที่ใส่เข้าไปก็จะทับกับข้อมูลเดิมอยู่ดี แล้วจะทราบได้อย่างไรว่าต้องใช้เซกเมนต์ใดที่จะไม่เกิดเหตุการณ์เช่นนี้ขึ้นจะต้องเป็นการใช้อย่างคุ้มค่าด้วย เขาจะทำได้หรือไม่ หรือเขาจะยอมแห้วต่อไป โปรดติดตาม..ย่อหน้าถัดไป

ผมจะใช้ค่าลอจิกคอลแอดเดรสจากของเดิมคือ 6547H:0000H ในหนึ่งเซกเมนต์มีขนาด 64 กิโลไบต์ หรือ 10000H ถ้าผมต้องการหาค่าลอจิกคอลแอดเดรสใหม่ที่ถัดจากเซกเมนต์เดิมคือ 6547H:FFFFH อยู่ 1 ไบต์ผมจะต้องเปลี่ยนให้อยู่ในรูปของฟิสิคอลแอดเดรสก่อนคือ

65470H+FFFFH=7546FH
บวกเพิ่มอีก 1 ไบต์
7546FH+1=75470H
เปลี่ยนกลับไปอยู่ในรูปของ
ลอจิกคอลแอดเดรสคือ 7547H:0000H

ทีนี้ผมก็จะได้เซกเมนต์ใหม่ที่ไม่ทับกับของเก่าแล้ว ถ้าคุณไม่เชื่อลองใช้

ค่าเซกเมนต์เดิมคือ 6547H แล้วลองเปลี่ยนค่าออฟเซตจาก 0000H ไปจนถึง 0FFFFH เพื่อคำนวณให้อยู่ในรูปฟิสิคอลแอดเดรสดูก็ได้ว่ามีออฟเซตใดที่ให้ค่าฟิสิคอลแอดเดรสเป็น 75470H (ล้อเล่นละครับ ใครจะไปทำ ยกเว้นแต่ท่านจะเขียนโปรแกรมคำนวณเอาไว้ดูเล่นก็ได้)

เขียนโปรแกรม

ผมขอให้ท่านลองพิจารณาการเพิ่มขึ้นของแอดเดรสจากเดิมเป็นค่าใหม่ โดยที่ผมจะใช้ค่าลอจิกแอดเดรสเป็น 298AH:0006H ซึ่งมีค่าฟิสิคอลแอดเดรสเป็น 298A6H ดังนี้

พอจะเห็นอะไรที่เปลี่ยนไปของเซกเมนต์บ้างไหมครับ ถ้าออฟเซตเปลี่ยนไป 16 ไบต์ เท่ากับว่าเราเพิ่มเซกเมนต์ขึ้นอีก 1 โดยที่ให้ออฟเซตเท่าเดิม จะให้ค่าแอดเดรสที่เดียวกัน จากสิ่งที่เห็นนี้เองเราจึงนำไปเขียนเป็นโปรแกรมที่จะช่วยให้เราสามารถใช้งานเมมโมรีได้ในกรณีที่ข้อมูลของเรามากเกินเซตเมนต์ที่ใช้อยู่

โปรแกรมนี้เป็นภาษาแอสเซมบลีที่จะต้องการลอจิกคอลแอดเดรสเริ่มต้นทาง DS:SI และค่าที่ต้องการบวกเพิ่มทาง BX และคืนค่าลอจิกคอลแอดเดรสที่บวกเพิ่มแล้วมาทาง DS:SI ดังนี้

```
; DS:SI - address , BX - Add value,
return DS:SI
Add_memory proc   near
           mov    ax,si
           mov    cl,4
           shr    ax,cl
           mov    cx,ds
           add    cx,ax
           mov    ax,bx
           and    si,OFh
           and    ax,OFh
           and    si,ax
           mov    ds,cx
           mov    cl,4
           shr    bx,cl
           mov    cx,ds
           add    cx,bx
           mov    ds,cx
           ret
Add_memory endp
```

โปรแกรมนี้จะให้ค่าลอจิกคอลแอดเดรสที่มีค่า offset ที่น้อย ๆ ซึ่งจะให้ค่าในการใช้งานที่เซกเมนต์นั้นได้มากยิ่งขึ้น แต่โปรแกรมนี้บวกค่าแอดเดรสได้ไม่เกินค่าสูงสุดที่ BX ให้ได้คือ 64 กิโลไบต์ มันใช้ได้ในกรณีที่ออฟเซตเพิ่มจนกระทั่งใกล้ OFFFFH แล้วท่านต้องการให้บวกเพิ่มไปอีก ซึ่งอาจจะทำให้ค่าในออฟเซตวนกลับไปตั้งต้นใหม่ที่ 0000H

ผมมีอีกโปรแกรมหนึ่งที่ใช้งานคล้าย ๆ กัน แต่เป็นภาษาซีใช้ในการบวกตัวแปรแบบ far pointer (ท่านที่ยังไม่ค่อยรู้เรื่องพอยน์เตอร์ให้ลองค้นหาอ่านเองเองนะครับหรือถ้าผมมีโอกาสคงได้เจอกันในเรื่องของพอยน์เตอร์) แต่สามารถบวกค่าได้มากกว่า 64 กิโลไบต์ แต่ยังไงก็ไม่เกิน 1 เมกะไบต์แน่นอน เนื่องจาก X86 มีเมมโมรีให้แค่ 1 เมกะไบต์เท่านั้น หรือถ้าท่านต้องการบวกก็ตามใจ แล้วถ้าเกิดผลข้างเคียงจากการกระทำครั้งนี้ ผมไม่รับผิดชอบใด ๆ ทั้งสิ้น

```
char far *New_pointer (char far
*point, long add_value)
unsigned int segment, offset;

  segment=FP_SEG (point);
  offset =FP_OFF (point);
  segment+=(offset/16);
  offset&=0x000F;
  offset+=(add_value/16L);
  return (MK_FP (segment,offset));
```

ฟังก์ชันนี้จะรีเทอร์นค่ากลับมาเป็น far pointer ให้นำไปใช้ได้ เนื่องจากการใช้ far pointer นั้น ภาษาซีไม่ได้บวกค่าของเซกเมนต์ในขณะที่ออฟเซตได้เกิน 64 กิโลไบต์ไปแล้ว ทำให้เกิดการวนรอบของข้อมูลขึ้นได้ แต่ภาษาซีได้เตรียม huge pointer ไว้เพื่อแก้ปัญหานี้เหมือนกัน แต่การทำงานของโปรแกรมที่ใช้ huge pointer จะช้าลง เนื่องจากจะต้องคอยเพิ่มค่าเซกเมนต์และออฟเซตในลักษณะของฟังก์ชันที่ผมยกตัวอย่างให้ไปแม้ว่าจะเป็นการบวกค่าถัดไปเพียง 1 ไบต์ก็ตาม

ผมจะไม่กล่าวถึงว่าเมมโมรีในส่วนที่ต่ำกว่า 1 เมกะไบต์นั้น IBM PC ได้ใช้ส่วนไหนทำอะไรบ้าง เนื่องจากผมยังไม่เห็นว่าจะต้องใช้ในขณะนี้ และผมคิดว่าหลายท่านคงทราบดีอยู่แล้ว หรือบางท่านที่อ่านมาจนถึงที่นี่แล้วขอให้ท่านลองนำไปใช้ดูว่าจริงอย่างที่ผมเขียน (พิมพ์) หรือเปล่า อย่าเชื่อผมง่าย ๆ ผมอาจจะหลอกก็ได้ (ล้อเล่นนะครับ ผมเพียงแต่อยากเห็นท่านกล้า ๆ ใช้คอมพิวเตอร์ให้คุ้มค่าเท่านั้น) หรือท่านจะว่าผมขายของเก่าก็ไม่เป็นไร สวัสดีครับ

ค่าที่เพิ่มขึ้น	(ไบต์)	physical address ใหม่ที่ได้	logical address ใหม่
00001H	(1)	298A6H+00001H = 298A7H	298AH:0007H
00010H	(16)	298A6H+00010H = 298B6H	298BH:0006H
00014H	(20)	298A6H+00014H = 298BAH	298BH:000AH
00100H	(256)	298A6H+00100H = 299A6H	299AH:0006H
01000H	(4K)	298A6H+01000H = 2A8A6H	2A8AH:0006H
10000H	(64)	298A6H+10000H = 398A6H	398AH:0006H